Parallel Architectures

Prof. Michele Amoretti

High Performance Computing 2022/2023

Outline

- Definition and motivation
- System evolution
- Flynn’s taxonomy
- Memory and communication

Definition and motivation

Sequential computing

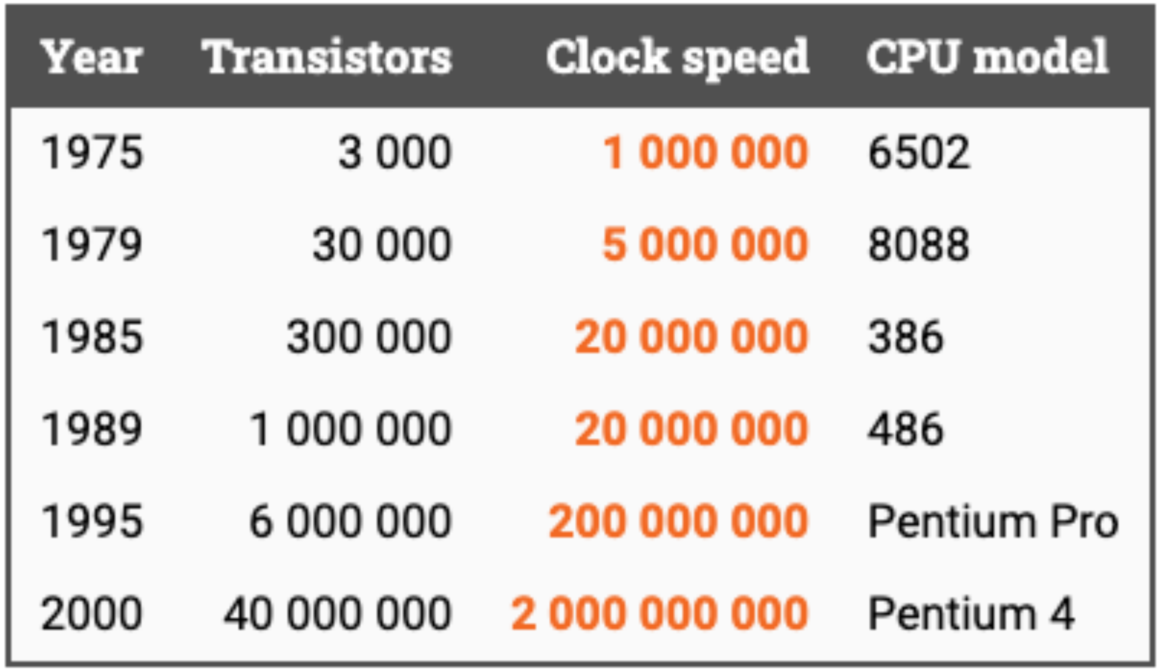
Traditionally, software has been written for sequential computers, i.e., to be executed on a machine provided with a unique CPU.

A sequential program is a sequence of instructions, that are executed one after the other.

Sequential computing

Until around 2000, we used to see the performance of CPUs rapidly increasing, primarily for two reasons.

First, the **clock speed** of the CPU (the number of clock cycles per second) increased:



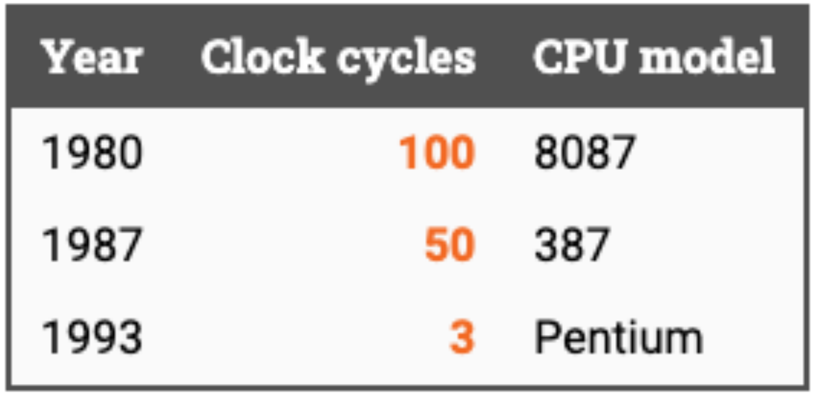
Year	Transistors	Clock speed	CPU model
1975	3 000	1 000 000	6502
1979	30 000	5 000 000	8088
1985	300 000	20 000 000	386
1989	1 000 000	20 000 000	486
1995	6 000 000	200 000 000	Pentium Pro
2000	40 000 000	2 000 000 000	Pentium 4

Sequential computing

Second, CPUs were using fewer and **fewer clock cycles** per operation.

- **Latency:** time to perform an operation from start to finish

As a simple example, consider the **FMUL** instruction that multiplies two 80-bit floating point numbers on Intel CPUs. The total latency of the instruction (roughly speaking, the time it takes to launch multiplication to the time when the result is available) has dropped significantly over the years:



Year	Clock cycles	CPU model
1980	100	8087
1987	50	387
1993	3	Pentium

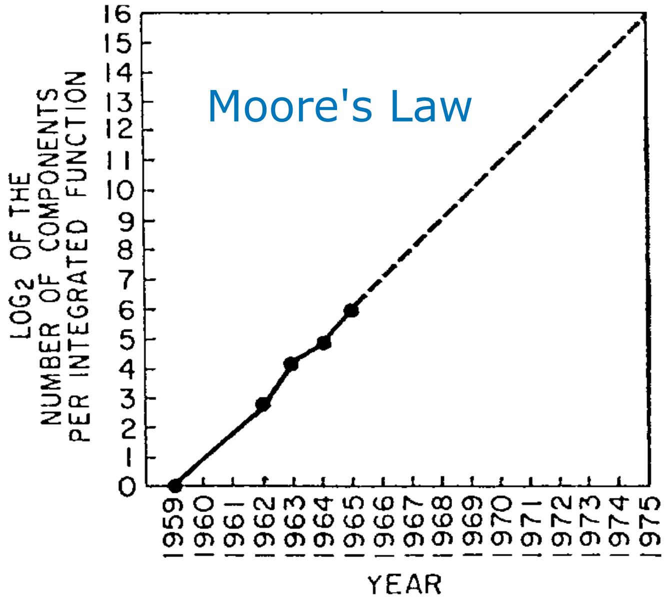
Sequential computing

The performance of a sequential processor depends on the **data transmission rate** inside hardware. Absolute limits are:

- light speed (30 cm/nanosec)
- copper lines transmission speed (9 cm/nanosec)

Another important issue is **power consumption** (→ heat generation).

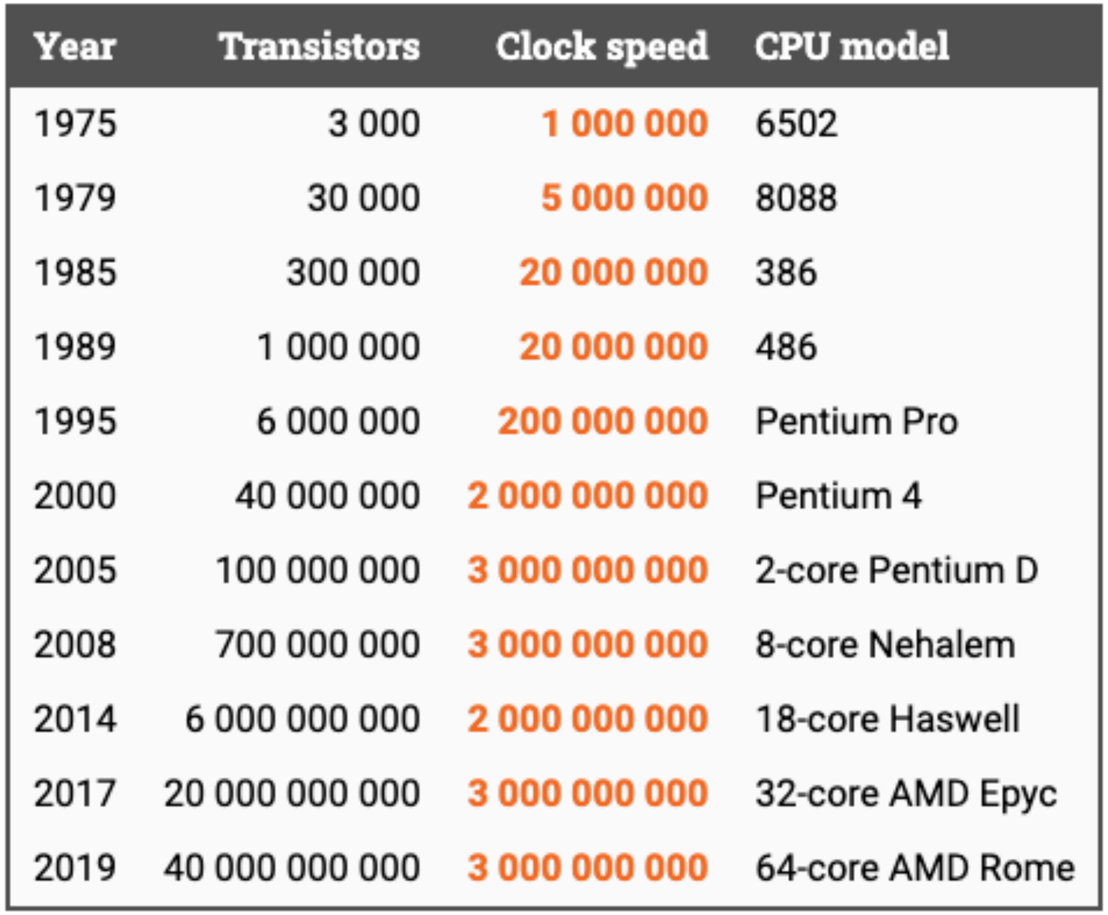
VLSI technologies allows to increase the number of devices on each chip, but also in this case there are limits over which miniaturization is not possible.



Parallel computing

Around year 2000, everything changed. Moore's law was still doing fine, and the number of transistors kept increasing.

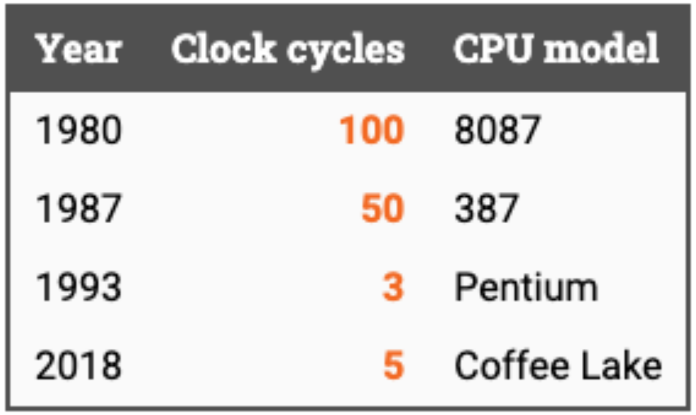
However, clock speeds stopped improving. The clock speed of a modern computer is almost always somewhere in the ballpark of 2–3 GHz:



Year	Transistors	Clock speed	CPU model
1975	3 000	1 000 000	6502
1979	30 000	5 000 000	8088
1985	300 000	20 000 000	386
1989	1 000 000	20 000 000	486
1995	6 000 000	200 000 000	Pentium Pro
2000	40 000 000	2 000 000 000	Pentium 4
2005	100 000 000	3 000 000 000	2-core Pentium D
2008	700 000 000	3 000 000 000	8-core Nehalem
2014	6 000 000 000	2 000 000 000	18-core Haswell
2017	20 000 000 000	3 000 000 000	32-core AMD Epyc
2019	40 000 000 000	3 000 000 000	64-core AMD Rome

Parallel computing

Instruction latencies have not improved much, either; sometimes the latencies are nowadays **worse** than what we saw twenty years ago. Here are examples of the latency of the **FMUL** operation:



Year	Clock cycles	CPU model
1980	100	8087
1987	50	387
1993	3	Pentium
2018	5	Coffee Lake

Since 2000, the number of transistors in a state-of-the-art CPU has increased by more than two orders of magnitude, but this is not visible in the performance if we look at the time it takes to complete a single instruction. A single floating point multiplication took roughly 2 nanoseconds in 2000, and it still takes roughly 2 nanoseconds today.

Parallel computing

So what is happening? And why are the CPU manufacturers packing more and more transistors in the CPUs if it does not seem to help with the performance?

All modern computers have **massively parallel processors**. The **CPU** of a normal desktop or laptop computer may easily have **hundreds** of arithmetic operations in progress simultaneously:

- there are multiple **CPU cores**
- each core has multiple **execution units** that can be used simultaneously
- each execution unit can perform **wide vector operations**, an
- the execution units are **pipelined** so one does not need to wait for the previous instruction to finish before starting the next one.

Parallel computing

So performance of modern CPUs is steadily increasing, but it is **new kind of performance**: the latency of individual operations is not improving at all, only throughput is improving.

- **Throughput:** how many operations are completed per time unit, in the long run

And to benefit from the new kind of performance, we need **new kind of software** that explicitly takes into account the difference between throughput and latency.

Applications

Parallelism is mainly employed in the context of **high performance computing** for scientific and industrial applications:

- Weather forecast and climate analysis
- Geological analysis
- Fluid dynamics simulation
- Chemical and nuclear reactions
- Biology
- Human genome sequencing and mapping
- Mechanical design
- VLSI electronic design
- Astrophysics
- Machine Learning
- Big Data Analysis

More than Moore

M. Mitchell Waldrop
(physicist, science writer)

"The industry road map released next month will for the first time lay out a research and development plan that is not centred on Moore's law. Instead, it will follow what might be called the ***More than Moore strategy****: rather than making the chips better and letting the applications follow, it will start with applications — from smartphones and supercomputers to data centres in the cloud — and work downwards to see what chips are needed to support them."*

NATURE, VOL 530, 11 FEBRUARY 2016

The top 500 supercomputers

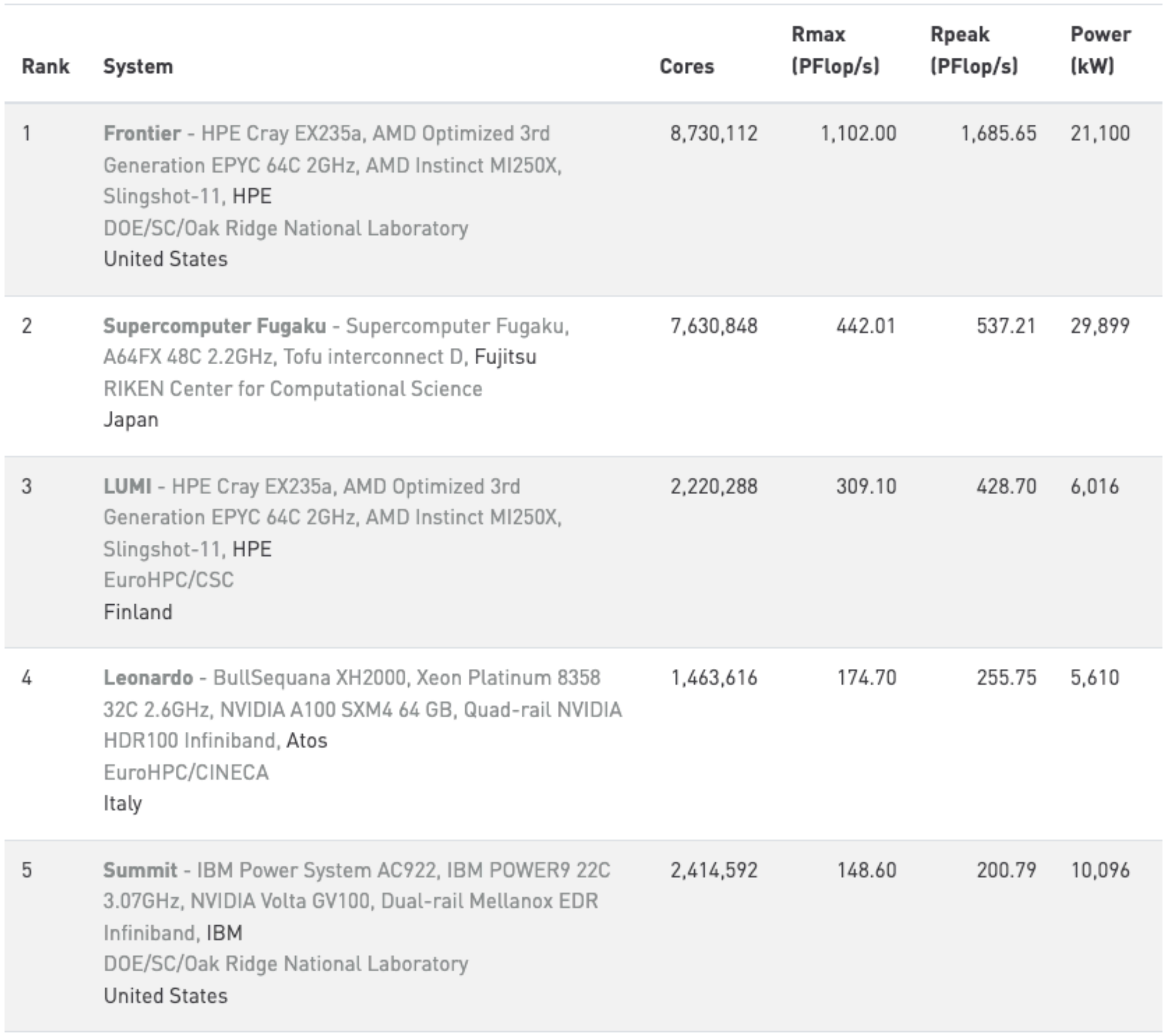
https://www.top500.org/

Computers are ranked by their performance on the *LINPACK benchmark* (problem: to solve a dense system of linear equations).

The adopted version of the benchmark allows the user to scale the size of the problem and to optimize the software in order to achieve the best performance for a given machine.

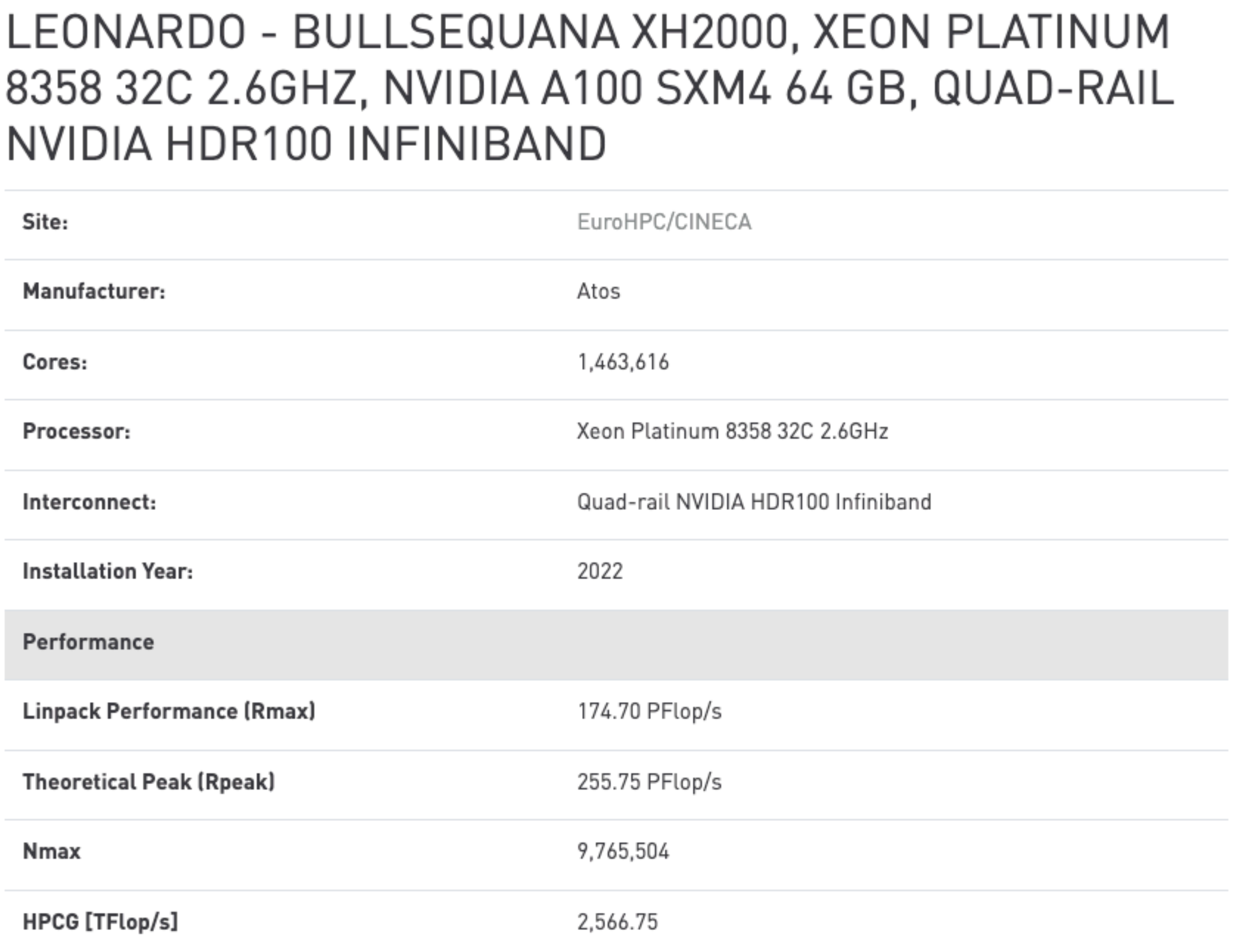
Since the problem is very regular, the performance achieved is quite high, and the performance numbers give a good representation of peak performance.

The top 5 supercomputers - November 2022



Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,463,616	174.70	255.75	5,610
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

The top 5 supercomputers - November 2022

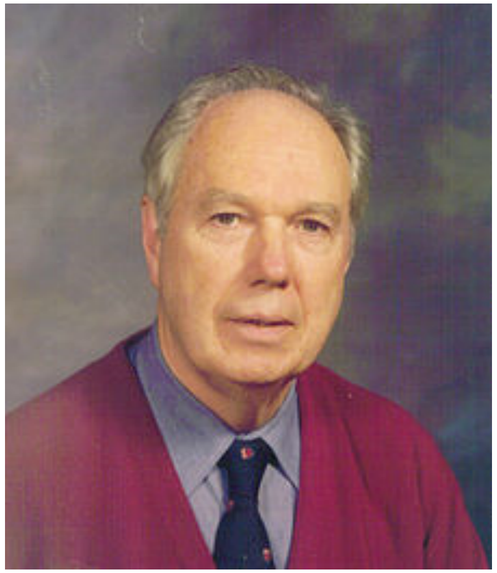


LEONARDO - BULLSEQUANA XH2000, XEON PLATINUM 8358 32C 2.6GHZ, NVIDIA A100 SXM4 64 GB, QUAD-RAIL NVIDIA HDR100 INFINIBAND

Site:	EuroHPC/CINECA
Manufacturer:	Atos
Cores:	1,463,616
Processor:	Xeon Platinum 8358 32C 2.6GHz
Interconnect:	Quad-rail NVIDIA HDR100 Infiniband
Installation Year:	2022
Performance	
Linpack Performance (Rmax)	174.70 PFlop/s
Theoretical Peak (Rpeak)	255.75 PFlop/s
Nmax	9,765,504
HPCG [TFlop/s]	2,566.75

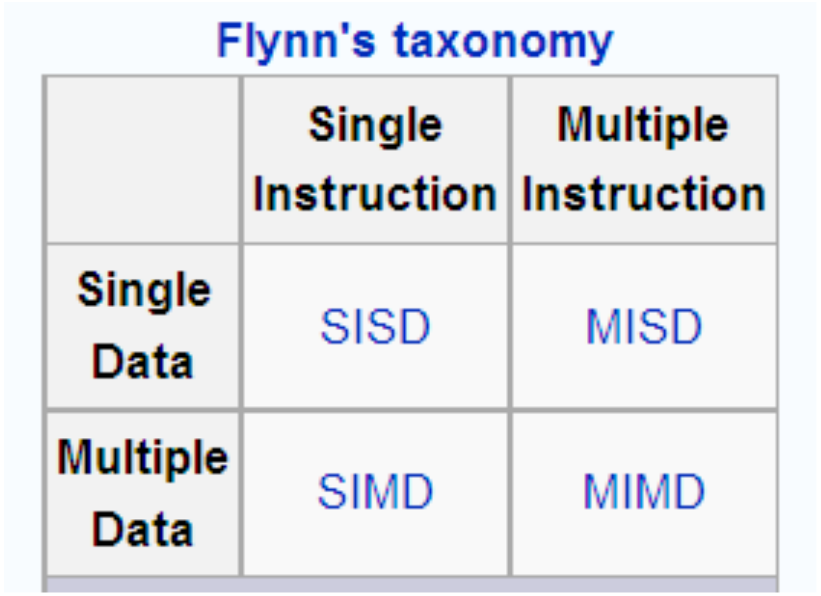
Flynn’s taxonomy

Flynn’s taxonomy



In 1966, Michael J. Flynn proposed one of the earliest classifications for parallel (and sequential) computers, now known as Flynn's taxonomy.

Flynn’s taxonomy separates architectures based on the realization of instruction and data flows.
Such flows can be single or multiple.



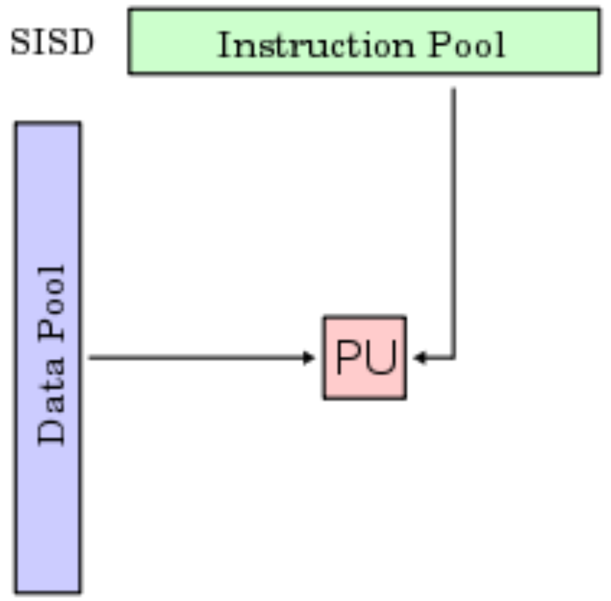
Flynn's taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

SISD

Single Instruction, Single Data

- One sequential processor (not parallel!)
- Single instruction: a flow of instructions is presented to the CPU and processed
- Single data: used as input in a clock cycle
- Deterministic execution
- It is the most traditional architecture



Examples:

- PCs
- workstations
- mainframes

before multiprocessor and multi-core architectures diffusion

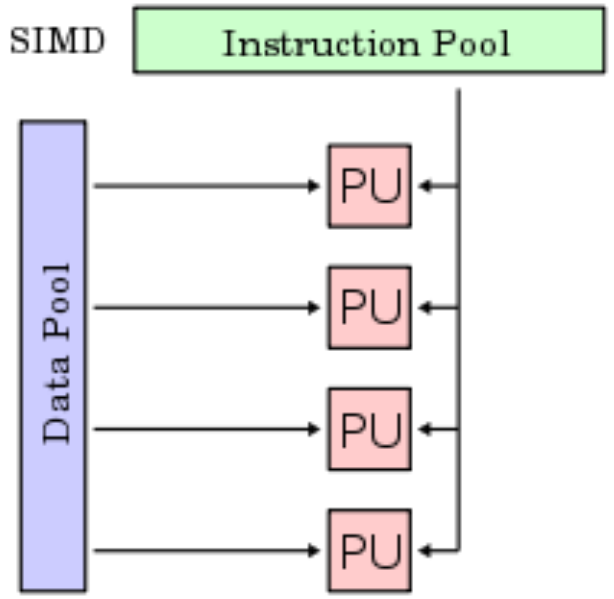
Superscalar architectures

A superscalar CPU architecture implements a form of parallelism called **instruction-level parallelism** within a single processor. It thereby allows faster CPU throughput than would otherwise be possible at the same clock rate.

A superscalar processor executes more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to redundant functional units of the CPU. Each functional unit is an execution resource within a single CPU such as an arithmetic logic unit, a bit shifter, or a multiplier.

Usually, superscalar CPUs exploit **pipelining**, which is another performance enhancement technique.

SIMD



Single Instruction, Multiple Data
(**data-level parallelism**)

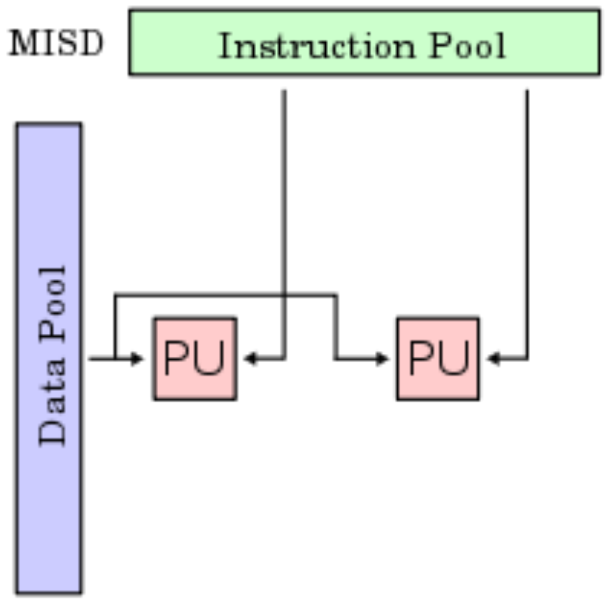
• Single instruction: a flow of instructions is presented to the CPU and processed
• Multiple data: each processor can operate on a different data element
• Synchronous, deterministic execution
• This type of machine usually has:
- an instruction distributor
- large bandwidth in the connection network
- a large number of not-so-powerful processors

Examples:
• Array Processor: Connection Machine CM-2, Maspar MP-1, MP-2
• Vector supercomputer: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2
• Most modern CPU designs include SIMD instructions

MISD

Multiple Instruction, Single Data

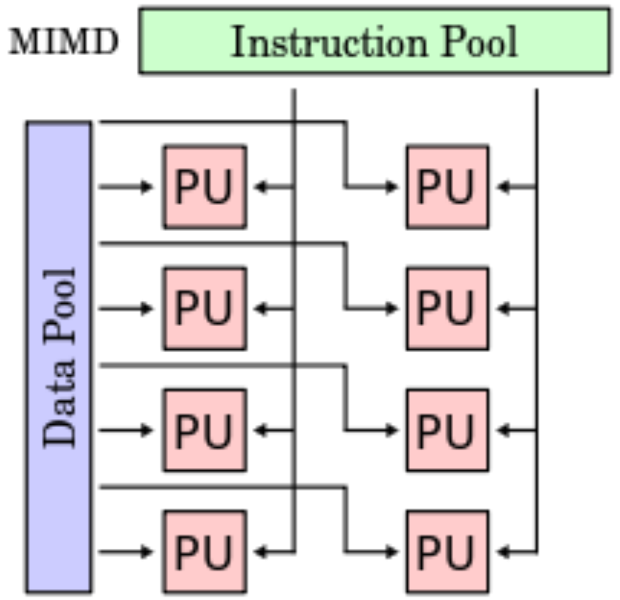
- A unique data flow feeds many CPUs
- Each CPU processes the data flow independently from the others



Examples:
Not many instances of this architecture exist. An example is the C.mmp built by Carnegie-Mellon University (in 1971). However, this computer is reconfigurable and can operate in SIMD, MISD and MIMD modes.

Possible applications:
- Different filters on a unique signal flow
- Different algorithms to decode the same encoded message

MIMD



Multiple Instruction, Multiple Data
(**thread-level parallelism**)

• Most modern parallel computers fit this category
• Multiple instructions: each processor can execute a different instruction flow
• Multiple data: each processor may operate on a different data flow
• Execution can be synchronous or asynchronous, deterministic or non-deterministic

Examples:
• MIMD multiprocessors
• multi-core architectures (all processors are on the same chip)
• most of current supercomputers

Memory and communication

Basic concepts

Task – An element of the computational activity. A task is a program or a set of instructions that is executed by a processor.

Parallel Tasks – A set of tasks that can be executed by different processors at the same time, providing correct results (as if they were executed sequentially).

Sequential Execution – Execution of a set of instructions one after the other. All parallel tasks are characterized by parts that need to be executed sequentially.

Parallel Execution – Execution of a program by means of parallel tasks. Each task may contain the same set of instructions, as well as different sets.

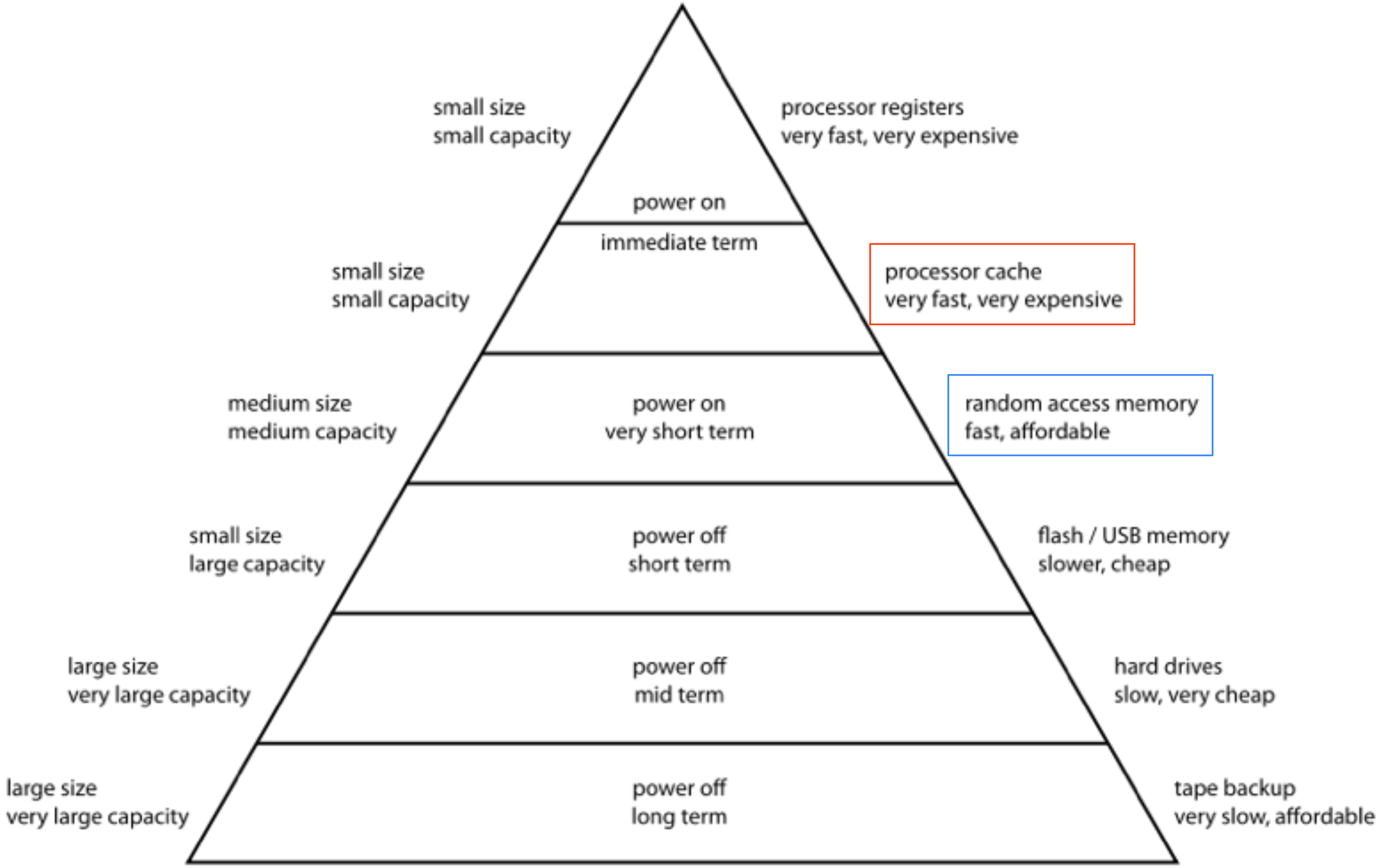
Synchronization

Parallel tasks usually need to exchange data. There are many strategies for doing it, for example by means of a **shared bus**, or by means of a **network infrastructure**.

Coordinating parallel activities is usually achieved by means of **synchronization points** within the application. A task cannot proceed until another task has reached a logically equivalent point.

Synchronization usually implies at least one task to wait, thus raising an **efficiency reduction** in the execution of the parallel application.

Memory hierarchy



Memory hierarchy

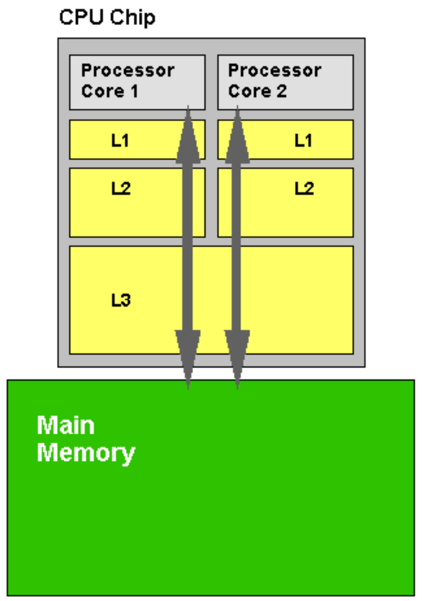
CPU caches are small pools of memory that store information the CPU is most likely to need next. Which information is loaded into cache depends on sophisticated algorithms and certain assumptions about programming code.

The goal of the cache system is to ensure that the CPU has the next bit of data it will need already loaded into cache by the time it goes looking for it (also called a **cache hit**).

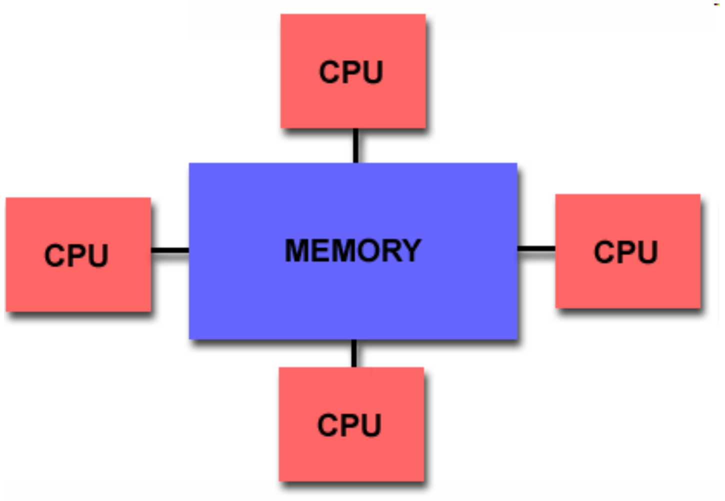
The closer the cache is to the CPU, the faster it is and the smaller it is.

L1 cache is small and very fast, and right next to the core that uses it.

L2 is bigger and slower, and still only used by a single core.
L3 is more common with modern multi-core machines, and is bigger again, slower again, and shared across cores on a single socket.



Shared Memory



In parallel computers with shared memory, all processors see the same global address space.

Processors may act independently, but share the same memory resources.
If a processor modifies data in memory, all other processors see it.

The corresponding programming model dictates that all tasks have the same "view" of the memory and are allowed to address the same "logical" locations, independently on where the memory is physically located.

Shared Memory

Pros

• Having a unique address space makes programming more easy.
• Data sharing among tasks is more fast and uniform, because of the proximity of processors.

Cons

• Low scaling. An increase of the number of CPUs and memories produces a geometric growth of traffic on the communication channels between CPUs and memories, on the system that controls the coherence of the cache and on the management of the cache itself.
• The programmer has to manage the synchronization mechanisms to allow correct access to data.

Shared Memory

A **symmetric multiprocessor (SMP)** is a computer system with multiple identical processors on the same motherboard, all sharing the same memory.

Because of the small size of the processors and the significant reduction in the requirements for bus bandwidth achieved by large caches, such symmetric multiprocessors are extremely cost-effective, provided that a sufficient amount of memory bandwidth exists.

Example: Quad-CPU AMD Opteron

Problems:
- complex to set up and maintain
- not scalable (because of bus contention)
- process migration can lead to poor cache utilization

Shared Memory

A **multi-core processor** has multiple execution units (*cores*) on the same chip, all sharing the same memory.

Multi-core processors are MIMD: different cores execute different threads (Multiple Instructions), operating on different parts of memory (Multiple Data).

Examples:

- IBM **Cell**, designed for use in the Sony Playstation 3
- Intel **Xeon**

Shared Memory

A **manycore processor** contains numerous (from a few tens to thousands) simple, independent processor cores.

They are very efficient for parallel computing, but have low single-thread performance.

Examples:

- Intel **Xeon Phi 7210 .. 7290F** (they have 64 to 72 cores)
http://ark.intel.com/products/codename/48999/Knights-Landing

- **CUDA** (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA for its most powerful GPUs.

Intel Xeon Phi Processor 7250 68c

Number of cores: 68

Number of threads: 272 (4x core)

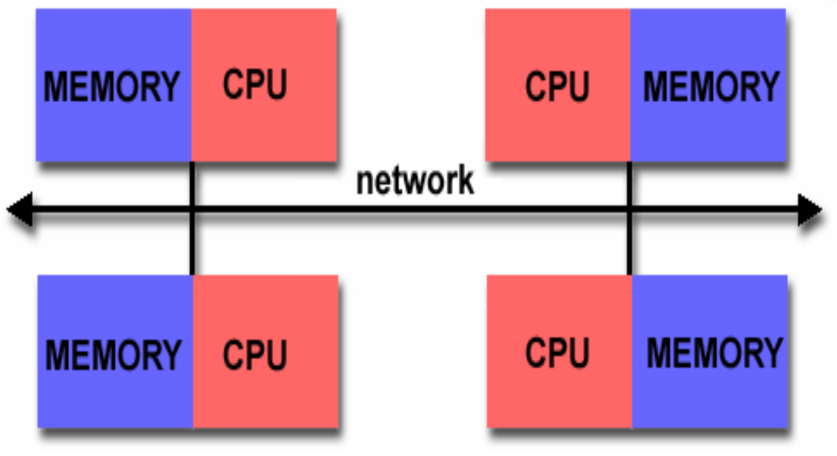
Processor base frequency: 1.4 GHz

L2 Cache: 34 MB

Max memory size: 384 GB

Peak DP compute: 3046 GFLOPS

Distributed Memory



Distributed memory requires a communication network for the information exchange.

Each processor has its own local memory. Each memory has a separate, independent address space.

Read/write operations are local, for which there are no problems of cache coherence. To allow a task to access remote data, the programmer must explicitly manage the communication among tasks.

The corresponding programming model is called **message passing**. It dictates that each task can directly access only its local memory and must use communicate with remote tasks to access remote data.

Distributed Memory

Pros

- Memory scales with the number of processors.
- Each processor can quickly access its memory, without interferences and overheads for cache coherence preservation.
- Cost effectiveness: it is possible to use common processors (commodity)

Cons

- The programmer is responsible of most details associated to the communication between processors.
- It may be difficult to effectively divide data in distributed memories.
- Access time to memories may not be uniform.

Distributed Memory

A **massively parallel processor (MPP)** is a single computer with many networked processors, each one being associated to its own private memory.

MPPs have many of the same characteristics as clusters, but MPPs have specialized high-speed interconnection networks (whereas clusters use commodity hardware for networking).

MPPs also tend to be larger than clusters.

Examples:

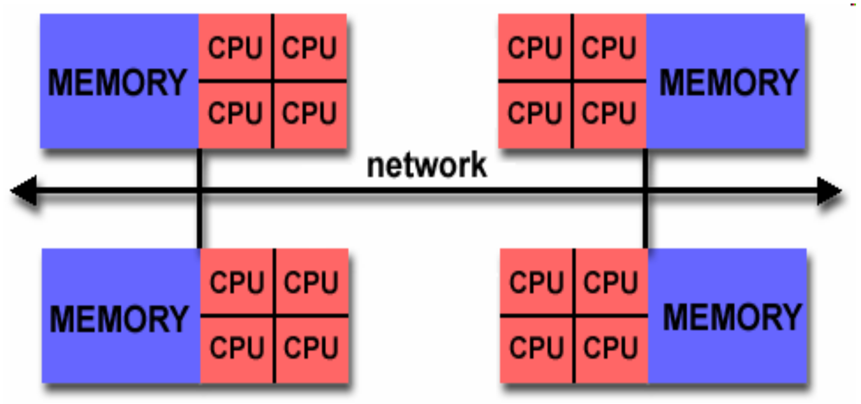
- Array Processor: Connection Machine CM-2, Maspar MP-1, MP-2
- Most modern supercomputers: IBM Blue Gene, etc. (see Top 500)

Mixed approach

Nowadays, most powerful computers use both shared and distributed memory.

Usually each node is a symmetric multiprocessor (SMP) with coherence control on the cache, or a multi-core system, or both.

Nodes are connected with a network infrastructure.



Network

The term **network topology** refers to the way in which a set of nodes are connected to each other.

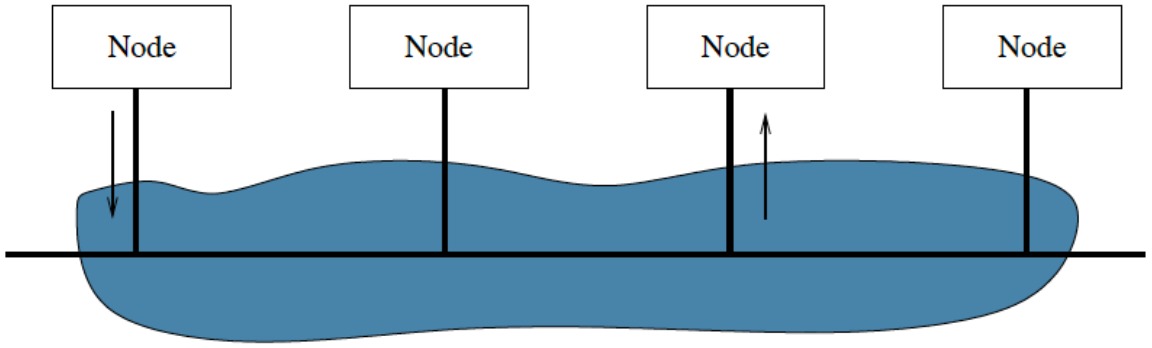
Network topologies arise in the context of
- parallel architectures
- parallel algorithms

Here we focus on actual, physical connections in parallel architectures.

An **interconnection network** is a system of links that connects one or more devices to each other for the purpose of inter-device communication.

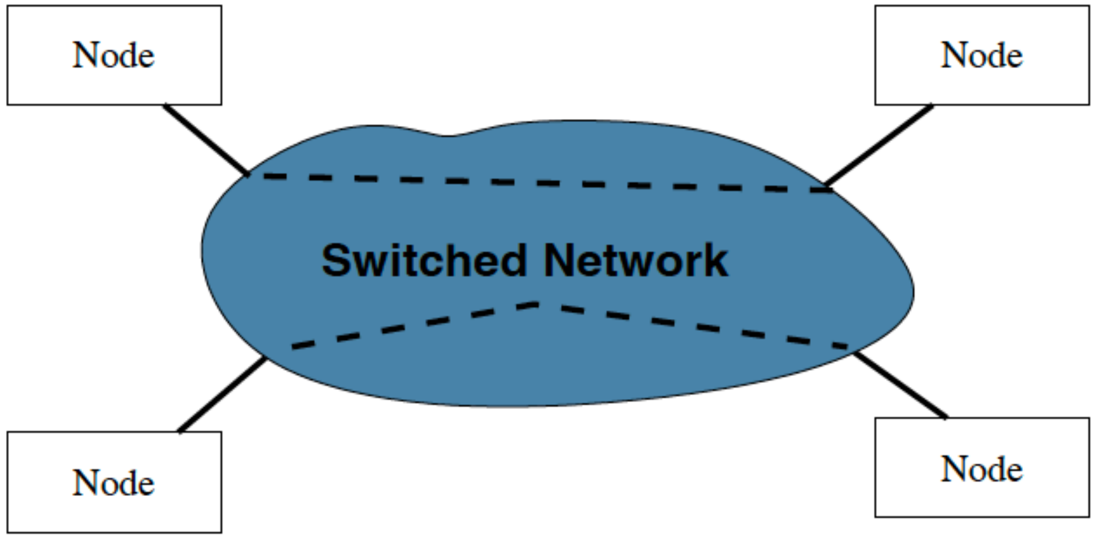
Network

A **shared network** can have at most one message on it at any time. For example, a **bus** is a shared network, as is traditional Ethernet.



Network

In contrast, a **switched network** allows point-to-point messages among pairs of nodes and therefore supports the transfer of multiple concurrent messages.

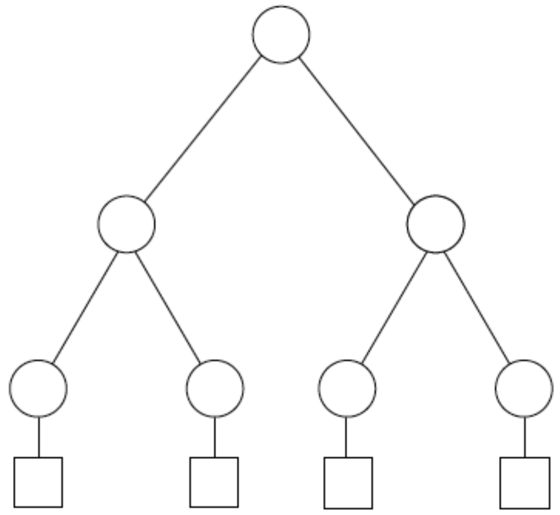


Interconnection network topologies

We use squares to represent processors and/or memories, and circles to represent switches.

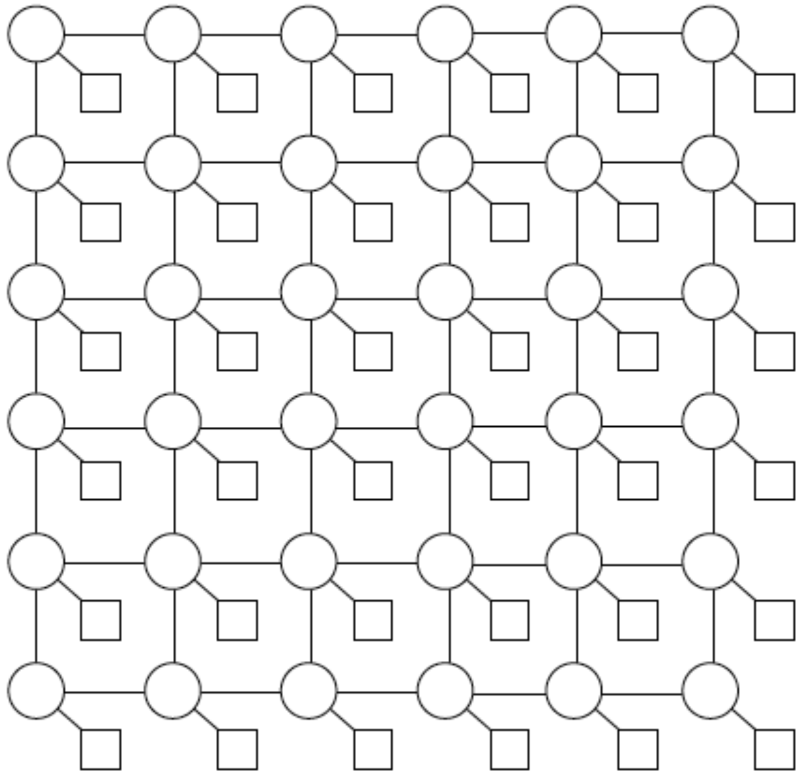
In a **direct** topology, there is exactly one switch for each processor node. In an **indirect** topology, the number of switches is greater than the number of processor nodes.

Binary trees are always indirect topologies, acting as a switching network to connect a bank of processors to each other.



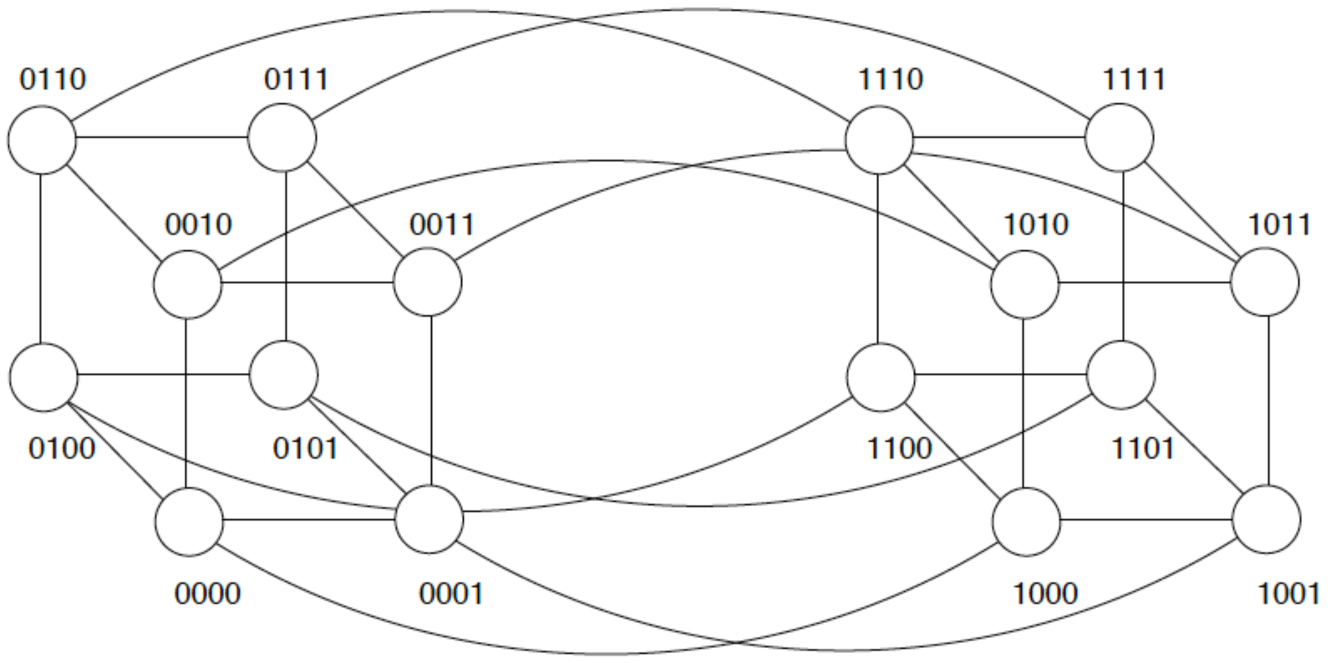
Interconnection network topologies

The **2D mesh** is almost always used as a direct topology, with a processor attached to each switch.



Interconnection network topologies

Hypercube networks are always direct topologies.



Network performance

Bandwidth (transmission rate) is the amount of data that can be transferred over a communication link per time unit.

Example: 10 Mbps = 10 megabits per second = 10^6 bits per second

Latency (delay) is the time that a data packet takes to travel from one point to another.

Latency becomes a problem only when real-time data transfer is necessary. Any latency beyond 200ms will give you problems in real-time communication.

References

• E. Aubanel, *Elements of Parallel Computing*, CRC Press, 2017

• M. Mitchell Waldrop, *More Than Moore*, Nature, Vol. 530, 11 February 2016

•*Top 500 Supercomputer Sites* http://www.top500.org

• M. Flynn*, Some Computer Organizations and Their Effectiveness,* IEEE Transactions on Computers, vol. C-21, 1972

• W. A. Wulf, C. G. Bell, *C.mmp - A multi-mini-processor*, Proc. AFIPS Conference, Vol. 41, Part II, 1972

• G. E. Blelloch, B. M. Maggs, *Parallel Algorithms*, The Computer Science Handbook, 2004